View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Elsevier - Publisher Connector

ELSEVIER

Discrete Applied Mathematics 54 (1994) 229–250

DISCRETE APPLIED MATHEMATICS

Regularity and locality in k-terminal graphs†

Sanjeev Mahajan, Joseph G. Peters*

School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada V5A 1S6

Received 18 May 1990; revised 21 May 1991

Abstract

In a 1985 paper, Bern, Lawler, and Wong described a general method for constructing algorithms to find an optimal subgraph in a given graph. When the given graph is a member of a *k-terminal recursive* family of graphs and is presented in the form of a *parse tree*, and when the optimal subgraph satisfies a property that is *regular* with respect to the family of graphs, then the method produces a linear-time algorithm. The algorithms assume the existence of *multiplication tables* that are specific to the regular property and to the family of graphs. In this paper we show that the general problem of computing these multiplication tables is unsolvable and provide a "pumping" lemma for proving that particular properties are not regular for particular k-terminal families. In contrast with these negative results, we show that all *local* properties, that can be verified by examining a bounded neighbourhood of each vertex in a graph, are regular with respect to all k-terminal recursive families of graphs, and we show how to automate the construction of the multiplication tables for any local property.

Key words: Linear-time algorithms; Regular graph properties; Local properties; k-terminal recursive graphs

1. Introduction

Combinatorial optimization problems that ask for an optimal subgraph or a vertex partition of a given weighted or unweighted graph arise in many settings. Most problems of this form are NP-hard for arbitrary graphs, but many can be solved quickly for restricted families of graphs. In fact, there are a large number of linear-time algorithms in the literature for such problems. Many of these linear-time algorithms are problem-specific and were developed using ad hoc techniques. Takamizawa et al. [14] appear to have been the first to introduce general techniques for constructing

†This research was supported by the Natural Sciences and Engineering Research Council of Canada under Grant No. A-0322.

*Corresponding author.

0166-218X/94/$07.00 © 1994—Elsevier Science B.V. All rights reserved

SSDI 0166-218X(93)E0103-6

linear-time algorithms for graph optimization problems. They used dynamic programming algorithms to solve a variety of optimal subgraph problems for series–parallel graphs. A number of researchers then began developing more general methodologies that applied to larger classes of graphs and problems [3, 5–9, 12, 13, 15, 16]. This paper is based on the methodology described by Bern et al. [6], and extended by Mahajan and Peters [12]. Wimer et al. [16] and Arnborg [1] provide good introductions and a more detailed history of the early development of the theory of linear-time graph algorithms and Hedetniemi [10] has compiled an extensive bibliography on the subject.

Intuitively, a family of graphs is *recursive* if every graph of the family can be generated by a finite number of applications of *composition operations* starting with a finite set of *basis* graphs. If each graph in a recursive family of graphs has a set of k distinguished vertices called *terminals*, and each composition operation is defined in terms of certain primitive operations on terminals, then the family of graphs is *k-terminal recursive*. The k-terminal recursive families of graphs include trees, series–parallel graphs, outerplanar graphs, and partial k-trees. Also, every k-terminal recursive family of graphs has a fixed upper bound on the *treewidth* of the graphs in the family, and graphs with treewidth at most k for fixed k are partial k-trees (see [15]).

Bern et al. define *regular properties* in terms of finite-range homomorphisms on graph × subgraph pairs (G, H) where G is a member of the k-terminal recursive family and H is a subgraph of G. The idea is to extend the composition operations from graphs to equivalence classes of graph × subgraph pairs. A property P is *regular* with respect to a family of graphs if the number of equivalence classes induced by each extended composition operator is finite (and therefore can be represented as a finite binary multiplication table).

Any member of a k-terminal recursive family of graphs can be represented as a *parse tree* in which the leaves are basis graphs and the internal nodes represent applications of composition operations to the subgraphs represented by their children. A dynamic programming algorithm can now be used to find an optimal subgraph satisfying a regular property P in any graph of the family by using the multiplication tables for P to work up from the leaves to the root of the parse tree of the graph. The algorithm is linear in the size of the parse tree which is linear in the size of the graph when the graph is a member of a k-terminal recursive family.

A disadvantage of the approach in [6] is that it is nonconstructive in two ways. Firstly, the linear-time algorithms assume the existence of *multiplication tables* (based on homomorphisms) which are specific to the property and to the family of graphs. Secondly, the algorithms require that the input be given in terms of a *parse tree*. The second issue has been investigated by Arnborg et al. [2], and by Arnborg and Proskurowski [4, 5]. In this paper, we will assume the existence of a parse tree for the input and will concentrate on the computational aspects of regularity. We start by proving that the problem of computing the multiplication tables of an arbitrary subgraph property for a fixed k-terminal recursive family of graphs is unsolvable. Then we develop a "pumping" lemma to establish the nonregularity of particular

graph properties with respect to particular k-terminal recursive families, and use it to prove the existence of properties that are regular with respect to some k-terminal recursive families and nonregular with respect to other families.

Motivated by the computational difficulties of dealing with arbitrary properties, we investigate *local* properties which can be verified by examining a bounded neighbourhood of each vertex of a graph. Our goal is to provide a simple graph theoretic heuristic for testing for regularity. The main contributions of this paper are proofs that all local properties are *uniformly regular*, that is, regular for all k-terminal recursive families of graphs, and an effective procedure for constructing the multiplication tables for any local property and any k-terminal family. Our construction procedure is based on finite state automata and provides an alternative to the logic-based methods of Arnborg et al. [3] and Borie et al. [8], and the algebraic approach of Bodlaender [7].

While we will not prove it here, it is easy to show (with the pumping lemma) that any attempt to generalize our definition of locality by replacing the finite state automata with more powerful machines gives nonregular properties. Thus, our notion of locality is the most general possible consistent with uniform regularity. Our notion of locality is not a complete characterization of uniform regularity, however, since there are nonlocal uniformly regular properties. Our approach could be generalized to obtain a more complete characterization, but the simplicity of our locality heuristic would be sacrificed in the process. Recent, logic-based approaches [3, 8, 9] come closer to a complete characterization by showing that properties describable in certain logic languages are uniformly regular. We discuss the generality of locality more fully in the last section of the paper.

2. Preliminaries

Definition 2.1. Let $G = (V, E, T)$ be a graph[1] with vertex set V, edge set E, and an ordered list T of k *terminals* chosen from V for some fixed integer k. (Note that the elements of T are not necessarily distinct.) Vertices in $V - T$ are called *internal* vertices. A *k-terminal recursive* family of graphs Γ is defined as follows.

(1) Let $B = \{B_1, B_2, \ldots, B_l\}$ be a finite set of basis graphs, where each B_i is a finite graph having an ordered list of k (not necessarily distinct) terminals (i.e., vertices).

(2) Let $O = \{\circ_1, \circ_2, \ldots, \circ_m\}$ be a finite set of binary rules of composition, whereby two graphs $G_i = (V_i, E_i, T_i)$ and $G_j = (V_j, E_j, T_j)$ can be combined to produce new graphs $G_{irj} = G_i \circ_r G_j$; $1 \leqslant r \leqslant m$. Each rule of composition $\circ_r$ consists of three suboperations on the terminals T_i and T_j:

   (i) Choose a subset T_i' of distinct terminals from the list T_i and identify each $x \in T_i'$ with a unique $y \in T_j$. Let T_j' denote the subset of affected terminals from the list T_j.

[1] We will only consider simple graphs in this paper although some of the results also apply to multigraphs.

(ii) Add any subset of the edges $\{(x, y) \mid x \in \bar{T}_i', y \in \bar{T}_j'\}$ to G_{irj}, where $\bar{T}_i'$ is the subset of terminals in the list T_i but not in T_i', and $\bar{T}_j'$ is defined similarly.

(iii) Select an ordered list of k (not necessarily distinct) terminals from the lists T_i and T_j to be the terminals of G_{irj}.

(3) Now define the family Γ of graphs recursively as follows:

(i) Any $B_i \in B$ is in Γ.

(ii) If G_i and G_j are in Γ and $\circ_r$ is an operation in O, then the graph $G_{irj} = G_i \circ_r G_j$ is also in Γ.

Definition 2.2. Let Γ be a k-terminal recursive family of graphs. The *parse tree* of a graph $G \in \Gamma$ is a tree in which the leaves correspond to the basis graphs from which G is constructed, and each internal node represents the result of applying a composition operation to the graphs represented by the subtrees rooted at its children. G_v is used to denote the subgraph of G corresponding to a node v of a parse tree. If H is a subgraph of G, then H_v is used to denote H restricted to G_v.

Example 2.3. The family of *rooted trees* is a 1-terminal recursive family of graphs defined as follows.

(1) The triple $(\{x\}, \{\ \}, (x))$ is a rooted tree with root (terminal) x.

(2) If $T_1 = (V_1, E_1, \{r_1\})$ and $T_2 = (V_2, E_2, \{r_2\})$ are rooted trees, then $T = (V_1 \cup V_2, E_1 \cup E_2 \cup \{(r_1, r_2)\}, \{r_1\})$ is a rooted tree (see Fig. 1).

Example 2.4. The family of *series–parallel graphs* is a 2-terminal recursive family defined as follows.

(1) The graph $G = (\{l, r\}, \{(l, r)\}, \{l, r\})$ is a series–parallel graph, with ordered list (l, r) of left and right terminals.

(2) If $G_1 = (V_1, E_1, (l_1, r_1))$ and $G_2 = (V_2, E_2, (l_2, r_2))$ are series–parallel graphs, then:

(a) The graph obtained by identifying r_1 and l_2 is a series–parallel graph, with left and right terminals l_1 and r_2. This graph is the *series* composition of G_1 and G_2.

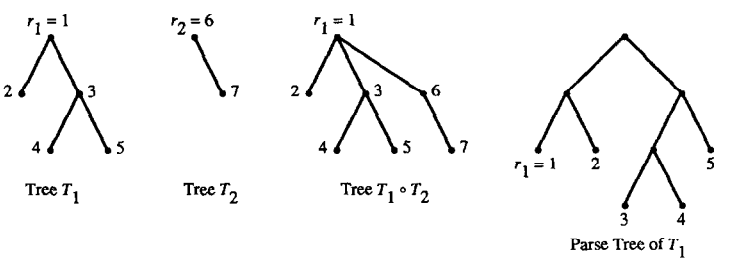

Fig. 1. Rooted tree composition operation.

(b) The series–parallel graph obtained by identifying l_1 and l_2 and also r_1 and r_2 is called the *parallel* composition of G_1 and G_2. The left and right terminals are $l_1 (= l_2)$ and $r_1 (= r_2)$.

Regular properties are defined in [6] in terms of homomorphisms. Let Γ be a k-terminal recursive family of graphs and let $\circ$ be a rule of composition of Γ. Suppose that P is a computable predicate defined on graph × subgraph pairs (G, H) where $G \in \Gamma$ and H is a subgraph of G. For now, we will restrict attention to subgraphs that are induced by vertex subsets. (We will remove the restriction in Section 5.) In this restricted case, we can extend the definition of $\circ$ to graph × subgraph pairs by defining $(G_1, H_1) \circ (G_2, H_2) = (G_1 \circ G_2, H_1 \cup H_2)$. A subgraph property P is *regular* if, for each composition operation $\circ$, the class of graph × subgraph pairs can be partitioned into a finite number of equivalence classes with respect to P. More formally, regularity is defined as follows.

Definition 2.5. Let Γ be a k-terminal recursive family of graphs and let $\Gamma^{\times}$ be the class of all *graph* × *subgraph pairs* (G, H) where $G \in \Gamma$ and H is a vertex-induced subgraph of G. A vertex of G is called *included* if it is also in H and is called *excluded* otherwise. Suppose that $\{\circ_1, \circ_2, \ldots, \circ_m\}$ is the set of composition operations of Γ extended to graph × subgraph pairs as follows: if $G_1 \circ_r G_2$ is defined, and no vertex of $G_1 \circ_r G_2$ is the result of identifying an included vertex with an excluded vertex, then $(G_1, H_1) \circ_r (G_2, H_2) = (G_1 \circ_r G_2, H_1 \cup H_2)$. Let P be a *subgraph property* (i.e., computable predicate) defined on graph × subgraph pairs where the subgraph is vertex-induced. We say that a pair (G, H) *satisfies* property P if H satisfies P in G. A *homomorphism* with respect to the class Γ and the property P is a partition of $\Gamma^{\times}$ into *equivalence classes* such that, for each operation $\circ_r$, the class to which a pair $(G, H) = (G_1, H_1) \circ_r (G_2, H_2)$ belongs is a binary function (denoted $\bullet_r$) of the classes to which (G_1, H_1) and (G_2, H_2) belong, and, for each equivalence class, either all or none of the members of the class satisfy P. Classes which satisfy P are *accepting classes* and classes which do not satisfy P are *rejecting classes*. P is *regular* with respect to Γ if it has a homomorphism with a finite number of equivalence classes (i.e., a *finite-range homomorphism*).

Since the number of equivalence classes is finite for a regular subgraph property, each binary operation $\bullet_i$ on the equivalence classes can be represented as a finite multiplication table. Bern et al. [6] describe a dynamic programming algorithm which uses the multiplication tables for a property and the parse tree of a graph to solve optimal subgraph problems. Starting with the leaves of the parse tree, the algorithm associates an *optimal representative* with each equivalence class at each node of the parse tree. Suppose that v is a node in the parse tree and that G_v is the subgraph that it represents. The optimal representative for each equivalence class at v will be a graph × subgraph pair (G_v, H) in the class for which H has optimum cardinality (or weight). At a leaf, G_v will be a basis graph B_i, and each optimal representative will be a pair of the form (B_i, H). At an interior node of the parse tree, the optimal representatives are functions of the optimal representatives of the children of v. For

each equivalence class C_k, consider all triples C_i, C_j, $\bullet_l$ such that $C_k = C_i \bullet_l C_j$, and choose the "best" such triple. The optimal solution is the best of the optimal representatives of the accepting classes at the root of the parse tree. Backtracking can be used to construct the optimal subgraph [6].

It is easy to show that this algorithm is linear in the *size* (number of nodes) of the parse tree. Furthermore, the size of the parse tree is linear in the size of the input for any k-terminal recursive family of graphs. Therefore, if the parse tree for any graph in a particular k-terminal recursive family of graphs can be obtained in linear time, then there is a linear-time algorithm for finding optimal subgraphs satisfying any given regular property P with respect to that k-terminal recursive family of graphs. We will restrict attention in this paper to unweighted graphs for clarity of presentation. To generalize all of our results to vertex-weighted problems, it is only necessary to make the obvious modifications to the dynamic programming algorithm.

3. Regular properties

The theory described in the previous section guarantees a linear-time algorithm for finding optimal subgraphs whenever the family of graphs is k-terminal recursive, the input consists of a parse tree of a graph, and the property is regular with respect to the family of graphs. A disadvantage of this theory is that it is not constructive; each linear-time algorithm must be constructed "by hand". The theory would be more useful if the construction of the linear-time algorithms could be automated. Ideally, we would like to have an algorithm which, given descriptions of a property, and of a k-terminal recursive family of graphs, produces a linear-time algorithm for finding an optimal subgraph obeying the property for any graph in the family. Unfortunately, this is impossible.

Theorem 3.1. *Given a fixed k-terminal recursive family of graphs Γ, and an arbitrary subgraph property P on graph × subgraph pairs, the problem of finding a finite-range homomorphism for P is unsolvable.*

Proof. It is known that the problem of determining whether $L(G) = \Sigma^*$ for an arbitrary context-free grammar G over an arbitrary finite alphabet Σ is undecidable [11]. Without loss of generality, we can assume that $\Sigma = \{a\}$. For any context-free grammar G, we can define P_G to be the property of graph × subgraph pairs on a given k-terminal recursive family of graphs Γ that is true for all graph × subgraph pairs on n vertices exactly when $a^n \in L$, and false otherwise. (Without loss of generality, assume that there exists a graph on n vertices which belongs to Γ for all n.) Clearly, P_G is true for all graph × subgraph pairs on Γ iff $L(G) = \Sigma^*$. Furthermore, the homomorphism with one equivalence class which is an accepting class is a finite-range homomorphism for P_G iff P_G is true for all graph × subgraph pairs. Suppose that there was an algorithm for finding a homomorphism for a property P_G. Then we could use the

method described in [6] to minimize the number of equivalence classes in such a homomorphism to obtain a homomorphism h. (The method in [6] is an adaptation of state minimization techniques for finite-state automata [11] to the problem of minimizing equivalence classes.) If h has one equivalence class which is an accepting class, then $L(G) = \Sigma^*$; otherwise, $L(G) \neq \Sigma^*$. Since the problem of determining whether $L(G) = \Sigma^*$ is undecidable, the problem of finding a finite-range homomorphism for a property P_G of graph × subgraph pairs on a given k-terminal recursive family of graphs is unsolvable. □

Theorem 3.1 shows that the general problem of finding homomorphisms for subgraph properties that can be specified in terms of pushdown automata is unsolvable. If we restrict attention further to properties that can be specified in terms of finite-state automata, the problem of finding homomorphisms is solvable, and most of the rest of this paper investigates the construction of homomorphisms (i.e., multiplication tables) for properties specified in terms of finite-state automata. First, however, we will develop a "pumping" lemma reminiscent of the pumping lemma for regular languages to establish the nonregularity of particular graph properties with respect to particular k-terminal recursive families of graphs. As an application of the pumping lemma, we will then show that a particular class of properties involving vertex independence is not regular with respect to the family of rooted trees.

Definition 3.2. A *path* from node u to node v in a parse tree of a graph G (where v is the ancestor of u in the parse tree) will be denoted $u \to v$. The length (number of edges) of $u \to v$ is denoted $|u \to v|$. A *fragment* corresponding to a path $u \to v$, denoted $frag(u \to v)$, is obtained by deleting all descendents of u (and their incident edges) from the parse subtree rooted at v (see Fig. 2). The *concatenation* of two fragments $frag(u \to v)$ and $frag(w \to x)$ is denoted $frag(u \to v)frag(w \to x)$ and is obtained by identifying v and w. The concatenation of $n \geqslant 0$ copies of $frag(u \to v)$ is denoted $frag^n(u \to v)$, and $frag^0(u \to v)$ is defined to be the trivial graph with one node. These notions can all be extended to graph × subgraph pairs (G, H) by considering all subgraph information to be inherited by all copies of a path or fragment. So, if a vertex or edge of a path or fragment is included (i.e., in H), then the corresponding vertices or edges in all copies of that path or fragment are also included.

Fig. 2 shows a tree T_1 and its parse tree with a fragment $frag(u \to v)$ indicated. The parse trees of T_2 and T_0 are obtained from the parse tree of T_1 by replacing $frag(u \to v)$ with $frag^2(u \to v)$ and $frag^0(u \to v)$, respectively.

Lemma 3.3. *Let P be a regular (vertex-induced) subgraph property with respect to a given k-terminal recursive family of graphs Γ. There exists a constant j, such that if (G, H) is a graph × subgraph pair in $\Gamma^\times$ which satisfies P, and the parse tree of G has a path $u \to v$ with $|u \to v| > j$, then $u \to v$ can be partitioned into three subpaths $u \to w$, $w \to x$, and $x \to v$, with $|w \to x| \geqslant 1$, and $|u \to x| \leqslant j$ such that the graph × subgraph pair*

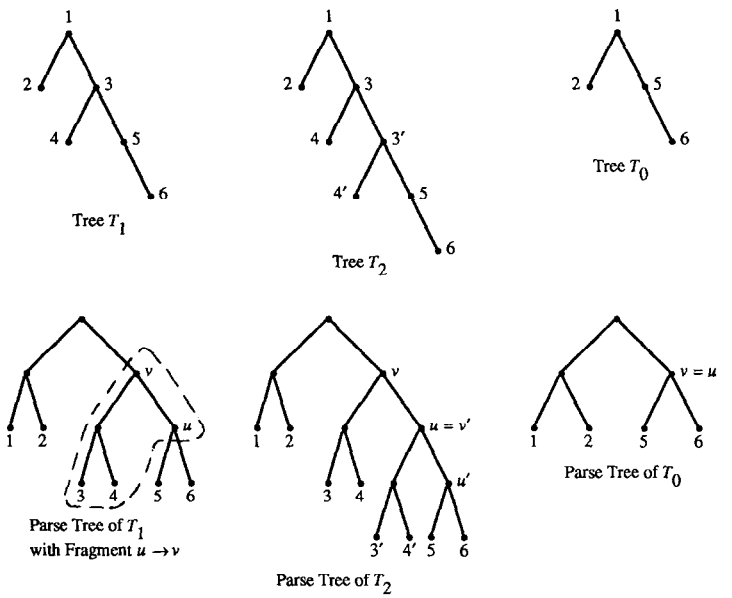

Fig. 2. Fragments and concatenation.

obtained by replacing $frag(u \to v)$ *by* $frag(u \to w)frag^m(w \to x)frag(x \to v)$ *also satisfies* P *for any* $m \geqslant 0$.

Proof. Choose j to be the number of equivalence classes in a finite-range homomorphism for P and Γ. Let (G, H) be any graph × subgraph pair in $\Gamma^{\times}$ which satisfies P, and which has a path $u \to v$ in its parse tree with $|u \to v| > j$. (G, H) satisfies P, so the equivalence class to which it belongs is an accepting class, say C_i. Since $|u \to v| > j$ and there are j equivalence classes, there must be two distinct nodes, w and x on $u \to v$ (where x is a proper ancestor of w) such that $|u \to x| \leqslant j$ and the equivalence classes of (G_w, H_w) and (G_x, H_x) are the same. Now, the equivalence class of (G_x, H_x) depends only on the classes of the children of x. So, for any ancestor y of x, the class of (G_y, H_y) does not change if $frag(w \to x)$ is replaced with $frag^n(w \to x)$ for any $n \geqslant 0$. Since the root of a parse tree is the ancestor of every node of the parse tree, the class of the root will still be C_i and the new graph × subgraph pair will satisfy P. □

Definition 3.4. Two vertices in a graph G are *f-independent* for some function $f: \mathbb{N} \to \mathbb{N}$ if the shortest path between them has at least $f(n)$ vertices, where n is the number of vertices of G. A graph × subgraph pair (G, H) is f-independent if every pair of vertices of H is f-independent in G.

Theorem 3.5. *Vertex f-independence is a nonregular property for rooted trees for any integer function* $f: \mathbb{N} \to \mathbb{N}$ *satisfying the following conditions.*

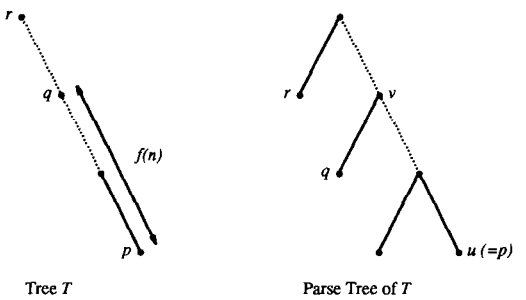

Fig. 3. A nonregular property.

(i) $f(n) \leqslant n$, $n \geqslant 1$,
(ii) $f(n) - f(n-1) \leqslant 1$, $n > 1$,
(iii) *for every nonnegative integer* j, *there exists an* $n > 1$ *such that* $f(n) > j$ *and* $f(n) - f(n-1) < 1$.

Proof. Assume to the contrary that vertex f-independence is regular, let j be the constant from Lemma 3.3, and choose n such that $f(n) > j$ and $f(n) - f(n-1) < 1$. Since $f(i) - f(i-1) \leqslant 1$ for all $i > 1$, we have $f(n-k) > f(n) - k$ for all $k < n$. Consider the graph × subgraph pair (G, H) of Fig. 3 where G is a simple path with n vertices, one leaf r is the root of G, and H consists of two vertices p and q which are distance $f(n)$ apart in G. In the corresponding parse tree of G, let u be the node corresponding to p and let v be the parent of the node corresponding to q. Clearly, $|u \to v| = f(n)$ and H is f-independent in G. Now, since f-independence is regular (by assumption), (G, H) is f-independent, and $|u \to v| = f(n) > j$, Lemma 3.3 guarantees that there are nodes w and x on $u \to v$ such that $|u \to x| \leqslant j$, $|w \to x| = k \geqslant 1$, and $frag(w \to x)$ can be replaced by $frag^0(w \to x)$ to get a new pair (G', H') that is f-independent. G' has $n - |w \to x| = n - k$ vertices and the two included vertices p and q are distance $f(n) - k < f(n-k)$ apart, so (G', H') is not f-independent. This is a contradiction, so f-independence is not a regular property for rooted trees. □

Example 3.6. The function $f(n) = \lceil \log n \rceil$ satisfies the three conditions of Theorem 3.5. (A suitable choice of n for condition (iii) is $n = 2^{j+2}$.) Therefore, $\lceil \log \rceil$-independence is a nonregular property for rooted trees.

It is easy to construct properties that are regular with respect to one k-terminal recursive family and not regular with respect to another. For example, let P be the property that is true for a graph × subgraph pair (G, H) if G is a member of a k-terminal recursive family Γ that does not include any rooted trees and H is independent in G, or if G is a rooted tree and H is $\lceil \log \rceil$-independent in G. We have just shown that $\lceil \log \rceil$-independence is not regular for the family of rooted trees, but we

will show in the next section that ordinary independence is regular with respect to all k-terminal recursive families. In fact, we will show that j-independence for any constant j, and many other properties, are regular for all k-terminal recursive families, and any of these properties could be used in place of ordinary independence in this example.

4. Local properties and uniform regularity

In the last section we showed that there is no guarantee that a property is regular with respect to a particular k-terminal recursive family, even if it is known to be regular with respect to some other k-terminal recursive family. There are, however, properties that are regular with respect to all k-terminal recursive families. We call such properties *uniformly regular*. In this section we will concentrate on subgraph properties that are *local* in the sense that it can be determined whether a subgraph satisfies the property in a graph by examining a bounded neighbourhood of each vertex of the graph. We will prove that all local subgraph properties are uniformly regular, and we will give an effective procedure for constructing the multiplication tables for any local property and any k-terminal recursive family. In this section, we continue to restrict attention to properties involving vertex-induced subgraphs.

In the formal definition that follows, locality is defined in terms of *vertex properties* and finite state machines. Intuitively, a vertex property is *j-local* if a finite state machine can verify that the property holds for a vertex by examining its *j-neighbourhood* (i.e., the set of vertices at distance j or less). A property defined on graph × subgraph pairs is *local* if it can be defined in terms of a j-local vertex property (for some fixed j) and a finite state machine that verifies that a "correct" number of vertices of the graph satisfy the vertex property. All of the finite state machines used below are automata that recognize subsets of the nonnegative integers. We will abuse notation by using $L(M)$ to denote the subset of integers corresponding to the strings accepted by machine M rather than the set of strings (which represent integers in unary notation) accepted by M.

Definition 4.1. Let $G = (V, E)$ be a graph, let (G, H) be a graph × subgraph pair, and let $N(v)$ denote the 1-*neighbourhood* of vertex v (the set of vertices adjacent to v in G). For any *vertex property* ψ defined on the vertices of graph × subgraph pairs, and any set S of vertices, let $|\psi(S)|$ denote the number of vertices in set S for which ψ is true, and $|\bar{\psi}(S)|$ the number of vertices in set S for which ψ is false. A vertex property p is 0-*local* if the truth value of $p(v)$ depends only on whether v is included or excluded for every vertex $v \in V$. (That is, all included vertices have the same truth value and all excluded vertices have the same truth value.) A *vertex property* p is *j-local* for some positive integer j if there is a $j-1$-local vertex property ψ, and four finite state machines M_1, M_2, M_3, and M_4, such that, for every graph × subgraph pair (G, H) and every vertex $v \in V$, $p(v)$ is true iff either v is included, $|\psi(N(v))| \in L(M_1)$, and

$|\bar{\psi}(N(v))| \in L(M_2)$, or v is excluded, $|\psi(N(v))| \in L(M_3)$, and $|\bar{\psi}(N(v))| \in L(M_4)$. A subgraph property P defined on graph × subgraph pairs is j-local if there is a j-local vertex property p, and two finite state machines M_a and M_b such that $|p(V)| \in L(M_a)$ and $|\bar{p}(V)| \in L(M_b)$. A subgraph property is *local* if it is j-local for some nonnegative integer j.

Note that all of the finite state machines used to define a local subgraph property P can be combined into a single machine using well-known techniques [11]. However, the following examples and proof are easier to understand if we continue to consider the machines separately.

Example 4.2. An included vertex in a graph × subgraph pair is *independent* if none of its neighbours is included. A graph × subgraph pair (G, H) is *independent* iff all its included vertices are independent. The vertex independence property is 1-local. The automata for the associated 1-local vertex property accept the sets $L(M_1) = \{0\}$ and $L(M_2) = L(M_3) = L(M_4) = \mathbb{Z}_0^+$ (the set of nonnegative integers). The associated 0-local property is true for included vertices and false for excluded vertices. The automata for the vertex independence property accept the sets $L(M_a) = \mathbb{Z}_0^+$ and $L(M_b) = \{0\}$.

Example 4.3. An included vertex in a graph × subgraph pair *dominates* itself and all of its neighbours. A graph × subgraph pair is dominated if all of its vertices are dominated. Since an included vertex is always dominated, and an excluded vertex is dominated if there is an included vertex in its 1-neighbourhood, the vertex domination property is 1-local. The associated sets are $L(M_a) = \mathbb{Z}_0^+$, $L(M_b) = \{0\}$, $L(M_1) = L(M_2) = L(M_4) = \mathbb{Z}_0^+$, and $L(M_3) = \mathbb{Z}^+$ (the positive integers). The associated 0-local property is true for included vertices and false for excluded vertices.

Example 4.4. Irredundance is an example of a 2-local property. An included vertex in a graph × subgraph pair is *redundant* if the set of vertices that it dominates can be dominated by other included vertices. An included vertex v is *irredundant* if there is an excluded vertex w in the neighbourhood of v such that the only included vertex in the neighbourhood of w is v. A graph × subgraph pair is irredundant if all of its included vertices are irredundant. To check if an included vertex is irredundant we need to check the 2-neighbourhood of v. The associated sets are $L(M_a) = \mathbb{Z}_0^+$ and $L(M_b) = \{0\}$. The sets for the associated 2-local property are $L(M_1) = \mathbb{Z}^+$ and $L(M_2) = L(M_3) = L(M_4) = \mathbb{Z}_0^+$, and the sets for the associated 1-local property are $L(M_1) = L(M_2) = \{\ \}$, $L(M_3) = \{1\}$, and $L(M_4) = \mathbb{Z}_0^+$. The associated 0-local property is true for included vertices and false for excluded vertices.

Theorem 4.5. *All local vertex-induced subgraph properties are uniformly regular.*

Proof. Let Γ be a k-terminal recursive family of graphs, and let (G, H) be a graph × subgraph pair. Assume, for now, that all of the composition operations of

Γ are defined in the same way for every pair of graphs. To prove that a local subgraph property P is regular for Γ, we need to show that there is a finite-range homomorphism for P. We prove this by first showing that there is a finite state machine for any j-local vertex property and then building a finite state machine with states that correspond to the equivalence classes of a homomorphism for P.

Let p be a j-local vertex property for some $j \geq 0$. The states of a machine for p must store enough information about the structure of a graph × subgraph pair (G, H) to determine the truth value of $p(v)$ for a vertex v of G when (G, H) is involved in any number of composition operations. We will prove by induction that there is a machine for p with a finite number of states. We will call the states of this machine *p-states*. If p is a 0-local vertex property, then the only information that has to be recorded about a vertex is whether it is included or excluded, so only two p-states are needed. If $j > 0$, then let M_1, M_2, M_3, M_4, and ψ be the four automata and the $j-1$-local vertex property associated with p. Assume, by induction, that the number of accepting and nonaccepting states in a machine for ψ (which we call ψ-states) is finite.

To determine the truth value of $p(v)$ for a vertex v of a graph × subgraph pair (G, H), when (G, H) is involved in a composition operation, it is necessary to store the current ψ-state of v, whether v is an internal vertex of a terminal, and, if v is a terminal, which terminal it is (recall that the set of k terminals is ordered). It is also necessary to know $|\psi(N(v))|$ and $|\bar{\psi}(N(v))|$ because M_1, M_2, M_3, and M_4 require this information. However, this is not enough information about the ψ-states of the neighbours of v to determine the value of $p(v)$ after a composition operation because two vertices in different accepting (or nonaccepting) ψ-states might not both be in accepting (or nonaccepting) ψ-states after a composition operation. The essential information about ψ-states for a vertex v is the numbers of neighbours of v in each of the ψ-states. Since this is a finite amount of information, there is a finite state machine for p that uses its states to store this detailed information about ψ-states along with $|\psi(N(v))|$ and $|\bar{\psi}(N(v))|$. A p-state in this machine is an accepting p-state if it stores values $|\psi(N(v))|$ and $|\bar{\psi}(N(v))|$ such that $|\psi(N(v))| \in L(M_1)$ and $|\bar{\psi}(N(v))| \in L(M_2)$ if v is included, or $|\psi(N(v))| \in L(M_3)$ and $|\bar{\psi}(N(v))| \in L(M_4)$ if v is excluded. Otherwise, it is a rejecting p-state.

Now, let P be a j-local subgraph property for some $j \geq 0$ with associated automata M_a and M_b and associated j-local vertex property p. To determine the truth value of P for a graph × subgraph pair (G, H), it is necessary to record information about the p-state of each vertex in G. By an argument similar to the one in the previous paragraph, it is sufficient to store the number of vertices in each of the p-states and there is a finite state machine M for P that stores this information. The accepting states of M are those that store values of $|p(V)|$ and $|\bar{p}(V)|$ such that $|p(V)| \in L(M_a)$ and $|\bar{p}(V)| \in L(M_b)$. Since the states of M correspond exactly to the equivalence classes of a finite-range homomorphism, we have established that P is regular.

For some classes of k-terminal recursive graphs, the composition operations may be defined differently for some pairs of graphs than for others. In particular, some basis graphs in the class may have fewer than k vertices, so they cannot have k distinct

terminals. Since the number of graphs for which this can happen is finite, all such differences in the definition of the composition operations can be handled by the addition of a finite number of states of M (and therefore a finite number of equivalence classes to the homomorphism). Cases in which the result of the composition of two graphs is undefined (such as attempts to identify an included terminal with an excluded terminal) can be grouped into a special rejecting state (or class). □

In the previous section, we showed that there are functions f for which $f(n)$-independence is not a regular property for rooted trees. It is easy to see that these properties are also not j-local for any constant j. However, it follows immediately from Theorem 4.5 that a local restriction of $f(n)$-independence, called *j-independence* is uniformly regular for every $j \geqslant 1$.

Definition 4.6. Two vertices in a graph are *j-independent*, for $j \geqslant 1$, if the shortest path between them has more than j edges, and *j-dependent* otherwise. A graph × subgraph pair (G, H) is j-independent if its included vertices are pairwise j-independent in G. Ordinary independence is 1-independence.

Corollary 4.7. *j-independence is a uniformly regular subgraph property for any $j \geqslant 1$.*

Theorem 4.8. *There is an effective procedure to build the multiplication tables for any local subgraph property for any k-terminal recursive family of graphs.*

Proof. Let P be a local subgraph property, let Γ be a k-terminal recursive family of graphs, and let M be the finite state machine for P from Theorem 4.5. The multiplication table for each operation $\bullet_l$ (corresponding to the composition operation $\circ_l$ of Γ) can be generated directly from M as follows. Start by determining the equivalence class of each graph × subgraph pair (G, H) for which G is one of the basis graphs of Γ. These classes form the set of initial equivalence classes. The number of basis graphs is finite, each basis graph has a bounded number of vertices, and M has a finite number of states, so this process can be completed in finite time. Add the initial equivalence classes to a set T and to the multiplication table T_l for each operation $\bullet_l$. Now, for every pair of (not necessarily distinct) equivalence classes of T and each operation $\bullet_l$, use M and the definition of $\circ_l$ to determine the result of composing graph × subgraph pairs from these two classes, and store the result in T_l. If any of these compositions yields a new equivalence class, add it to T. Repeat this process until no more equivalence classes can be added to T. Accepting classes are easily identified using M. Compositions that involve illegal operations, such as an attempt to identify an included vertex with an excluded vertex, can be grouped into a rejecting class. □

Theorem 4.5 can be used to establish an upper bound on the number of equivalence classes for any local subgraph property in terms of the number of states in the automata. In most cases, the number of equivalence classes generated by the construction of Theorem 4.8 will be smaller than this upper bound because "unreachable"

classes are not added to the set T. Sometimes the number of equivalence classes can be reduced further by using the same equivalence class minimization algorithm from [6] that was used in the proof of Theorem 3.1. One important class of properties for which these equivalence class minimization techniques can give dramatic reductions is the class of local properties which are true only if *all* vertices satisfy the associated vertex property (i.e., $L(M_a) = \mathbb{Z}_0^+$ and $L(M_b) = \{0\}$). As an example, consider such a property P that is 1-local, and let p, M_1, M_2, M_3, and M_4 be the 1-local vertex property and automata associated with P. Let (G, H) be a graph × subgraph pair where G is a member of a k-terminal family Γ and assume that all composition operations of Γ are defined in the same way for every pair of graphs. Internal vertices of (G, H) cannot change p-states when the pair is involved in a composition operation because internal vertices cannot acquire new neighbours. Therefore, it is only necessary to store the p-states of the k terminals and whether or not all internal vertices satisfy p. Since any 0-local property has two states, the number of different p-states is $(s_1^2 s_2^2 + s_3^2 s_4^2)$ (where s_1, s_2, s_3, and s_4 are the numbers of states in M_1, M_2, M_3, and M_4). This gives a total of $(s_1^2 s_2^2 + s_3^2 s_4^2)^k + 1$ equivalence classes corresponding to all possible choices of p-states for the k terminals and one rejecting state. This is much smaller than the bound from Theorem 4.5 which could be as large as $2^{2(k+1)(s_1^2 s_2^2 + s_3^2 s_4^2)}$. It is often possible to reduce the number of states even further for specific properties such as those in the following example.

Example 4.9. After eliminating unnecessary equivalence classes, the homomorphism for the vertex independence property (for any k-terminal family for which all of the composition operations are defined in the same way for every pair of graphs) has at most $2^k + 1$ equivalence classes. 2^k of the equivalence classes are accepting classes that represent the 2^k possible ways of including or excluding the k terminals of a graph × subgraph pair. For the vertex domination property, at most $3^k + 1$ equivalence classes are needed. Each terminal of a graph × subgraph pair can be classified as included (and therefore dominated), excluded and dominated, or excluded and undominated, so there are 3^k equivalence classes corresponding to the 3^k ways of classifying the k terminals. Of these classes, 2^k are accepting classes in which all terminals (included and excluded) are dominated. The other $3^k - 2^k$ classes are nonaccepting classes which contain undominated terminals but no undominated internal vertices. Any graph × subgraph pair containing an undominated (excluded) internal vertex is grouped into the rejecting class. For the irredundance property which is 2-local, the required number of equivalence classes is at most $(2^k + 4)^k + 1$ [12].

5. Regular vertex partition problems

To this point, we have restricted attention to properties of graph × subgraph pairs in which the subgraph is induced by a vertex subset. In this section, we show how to

extend our results to problems which ask for a partition of the vertex set of a graph. We will concentrate on optimization problems which ask for the coarsest or finest partition satisfying some property. An example of such a problem is the chromatic number problem which asks for the coarsest partition of the vertices satisfying the property that no pair of adjacent vertices is in the same component of the partition. We will continue to restrict attention to unweighted graphs for clarity of presentation, but all of the results hold for vertex-weighted problems with only minor modifications.

Definition 5.1. Let $G = (V, E)$ be a graph and let P be a graph property which can be checked by examining the adjacency structure of a labelled graph. A *vertex partition problem* is the problem of finding a *labelling function* $f: V \to \{1, \ldots, |V|\}$ satisfying P for which the cardinality of the range of f, denoted $|f[V]|$, is maximized or minimized. If $0 \leqslant f(v) \leqslant l$ for all $v \in V$, for some fixed positive integer l, then f is a *bounded labelling* for G. A vertex partition problem restricted to bounded labellings is called a *bounded partition problem.*

Example 5.2. Let $G \in \Gamma$ where $G = (V, E)$ and Γ is a k-terminal family of graphs. A *colouring* of V is a labelling function f such that for every $v \in V(G)$, $f(v) \neq f(v')$ for every $v' \in N(v)$. The *vertex colouring problem* is the vertex partition problem of finding a colouring f for which $|f[V]|$ is minimized. To show that the vertex colouring problem is a bounded partition problem for any k-terminal recursive family Γ, first assume that every basis graph of Γ can be coloured with at most $2k$ colours. For any composition operation $\circ$ of Γ, if two graphs G_1 and G_2 are $2k$-colourable, then $G = G_1 \circ G_2$ is also $2k$-colourable because edges are only added between the $2k$ terminals of G_1 and G_2. It follows by induction on the number of composition operations used to build G that G is $2k$-colourable. If l is the maximum of the chromatic numbers of the basis graphs of Γ, and $m = \max\{l, 2k\}$, it follows immediately that every $G \in \Gamma$ is m-colourable. Since m is a constant independent of G, the vertex colouring problem is a bounded partition poblem for every k-terminal recursive family of graphs.

Example 5.3. A function $f: V \to \{1, \ldots, |V|\}$ is a *domatic labelling* of a graph $G = (V, E)$, if, for all i, the set $V_i = \{v \mid f(v) = i\}$ is a dominating set for G. The *domatic number problem* is the vertex partition problem of finding of domatic labelling f for which $|f[V]|$ is maximized. Suppose that $G \in \Gamma$ where Γ is a k-terminal recursive family. If $|V| = n$, then G has at most $(m + k^2)n$ edges, where m is the maximum number of edges in any basis graph of Γ. Let f be a domatic labelling of G with $|f[V]| = d$. Since each vertex v must be dominated by a vertex labelled i for each $1 \leqslant i \leqslant d$ and since v dominates itself, the number of edges in G is at least $n(d - 1)/2$. Therefore, $n(d - 1)/2 \leqslant (m + k^2)n$ which implies $d \leqslant 2(m + k^2) + 1$. Since d is bounded by a constant independent of G, the domatic number problem is a bounded partition problem for every k-terminal recursive family of graphs.

Definition 5.4. Let $f: D \to R$ be a function and let $S \subseteq D$. The range of f restricted to S is $f[S] = \{f(d) \mid d \in S\}$. The *union* of two functions $f_1: D_1 \to R_1$ and $f_2: D_2 \to R_2$ is the function $f_1 \oplus f_2: D_1 \cup D_2 \to R_1 \cup R_2$ defined by

$$f_1 \oplus f_2(d) = \begin{cases} f_1(d) & \text{if } d \in D_1 \setminus D_2, \\ f_2(d) & \text{if } d \in D_2 \setminus D_1, \\ f_1(d) = f_2(d) & \text{if } d \in D_1 \cap D_2. \end{cases}$$

We will generalize the definitions of *regular property* and *homomorphism* to bounded partition problems by considering graph × function pairs (G, f), where G is a member of a k-terminal recursive family of graphs Γ and f is a bounded labelling of G. The graph × subgraph pairs considered in previous sections are a special case of graph × function pairs in which the range of f is $\{0, 1\}$ (or $\{excluded, included\}$). When the definition of a rule of composition $\circ$ is extended to graph × function pairs in the natural way: $(G_1, f_1) \circ (G_2, f_2) = (G_1 \circ G_2, f_1 \oplus f_2)$, the corresponding definitions of *homomorphism* and *regular property* are essentially the same as their counterparts for graph × subgraph pairs.

Definition 5.5. Let Γ be a k-terminal recursive family of graphs and let $\Gamma^{\times}$ be the class of all *graph × function pairs* (G, f), where $G \in \Gamma$ and f is a bounded labelling of G. Suppose that $\{\circ_1, \circ_2, \ldots, \circ_m\}$, is the set of composition operations of Γ extended to graph × function pairs as follows: if $G_1 \circ_r G_2$ is defined, and no vertex of $G_1 \circ_r G_2$ is the result of identifying two vertices x and y with $f_1 \oplus f_2(x) \neq f_1 \oplus f_2(y)$, then $(G_1, f_1) \circ_r (G_2, f_2) = (G_1 \circ_r G_2, f_1 \oplus f_2)$. Let P be a *partition property* defined on graph × function pairs where the function is a bounded labelling. We say that a pair (G, f) *satisfies* property P if G satisfies P when it is labelled by f. A *homomorphism* with respect to the class Γ and the property P is a partition of $\Gamma^{\times}$ into *equivalence classes* such that, for each operation $\circ_r$, the class to which a pair $(G, f) = (G_1, f_1) \circ_r (G_2, f_2)$ belongs is a binary function (denoted $\bullet_r$) of the class to which (G_1, f_1) and (G_2, f_2) belong, and, for each equivalence class, either all or none of the members of the class satisfy P. Classes which satisfy P are *accepting classes* and classes which do not satisfy P are *rejecting classes*. P is *regular* with respect to Γ if it has a homomorphism with a finite number of equivalence classes.

Theorem 5.6. *If P is a regular partion property with respect to a k-terminal recursive family of graphs Γ, then there is a linear-time algorithm for finding an optimal bounded labelling that satisfies P for any $G \in \Gamma$ when G is given as a parse tree.*

Proof. The extension of the linear-time dynamic programming algorithm from graph × subgraph pairs to graph × function pairs is straightforward. The only difference is that the algorithm for graph × subgraph pairs partitions the vertices of a graph into two classes (included and excluded), whereas the algorithm for graph × function

pairs may produce more than two vertex classes (corresponding to the possible vertex labels). However, since a bounded labelling gives a constant number of vertex classes, the dynamic programming algorithm is still linear in the size of the input graph for any regular bounded partition property. □

The negative results in Section 3 extend easily to graph × functions pairs. The unsolvability of finding a finite-range homomorphism for an arbitrary partition property is immediate for graph × function pairs because graph × subgraph pairs are a special case of graph × function pairs. The proof of the pumping lemma requires only cosmetic changes to reflect the changes in the definition of regularity. It follows that there are partition properties that are not uniformly regular. However, the notion of locality also extends in a natural way to partition properties and, as was the case with graph × subgraph pairs, all local partition properties can be shown to be uniformly regular (for bounded partition problems). The main difference between the following definition and Definition 4.1 is that more finite state machines are needed to define vertex properties because there are l different vertex types instead of two (included and excluded), where l is the bound associated with the partition property.

Definition 5.7. Let $(G, f) \in \Gamma^{\times}$ where Γ is a k-terminal recursive family of graphs, $G = (V, E)$, and f is a bounded labelling of G with $|f[V]| \leqslant l$. Use $N(v)$ to denote the 1-*neighbourhood* of vertex v. For any *vertex property* ψ defined on the vertices of graph × function pairs, and any set S of vertices, let $|\psi_j(S)|$ denote the number of vertices in set S which are labelled j by f and for which ψ is true, and let $|\bar{\psi}_j(S)|$ denote the number of vertices in set S which are labelled j by f and for which ψ is false. That is, $|\psi_j(S)| = |\{v \in S \mid f(v) = j \text{ and } \psi(v) \text{ is true}\}|$ and $|\bar{\psi}_j(S)| = |\{v \in S \mid f(v) = j \text{ and } \psi(v) \text{ is false}\}|$. A vertex property p is 0-*local* if $f(v) = f(w)$ implies $p(v) = p(w)$ for all $v, w \in V$ (i.e., the truth value of $p(v)$ depends only on the value of $f(v)$). A *vertex property* p is j-*local* for some positive integer j if there is a $j-1$-local vertex property ψ, and $2l^2$ finite state machines M_{ij} and $\bar{M}_{ij}$, $1 \leqslant i, j \leqslant l$, such that, for every graph × function pair (G, f) and every vertex $v \in V$, $p(v)$ is true iff $f(v) = i$ and $|\psi_j(N(v))| \in L(M_{ij})$ and $|\bar{\psi}_j(N(v))| \in L(\bar{M}_{ij})$ for all $1 \leqslant j \leqslant l$ and any $1 \leqslant i \leqslant l$. A partition property P defined on graph × function pairs is j-local if there is a j-local vertex property p, and two finite state machines M_a and M_b such that $|p(V)| \in L(M_a)$ and $|\bar{p}(V)| \in L(M_b)$. A partition property is *local* if it is j-local for some nonnegative integer j.

Example 5.8. The partition property P associated with the vertex colouring problem is 1-local. The automata for the associated 1-local vertex property accept the sets $L(M_{ii}) = L(\bar{M}_{ii}) = L(\bar{M}_{ij}) = \{0\}$ and $L(M_{ij}) = \mathbb{Z}_0^+$ for $1 \leqslant i, j \leqslant l$ and $i \neq j$, where l is the bound from Example 5.2. The associated 0-local property is true for all vertices. The automata for P accept the sets $L(M_a) = \mathbb{Z}_0^+$ and $L(M_b) = \{0\}$. The property associated with the domatic number problem is also 1-local and the associated machines M_a and M_b and 0-local property are the same as for the vertex colouring problem. The automata for the associated 1-local vertex property accept the sets

$L(M_{ii}) = \mathbb{Z}_0^+$, $L(M_{ij}) = \mathbb{Z}^+$, and $L(\bar{M}_{ii}) = L(\bar{M}_{ij}) = \{0\}$ for $1 \leqslant i, j \leqslant l$ and $i \neq j$, where l is the bound from Example 5.3.

Theorem 5.9. *All local partition properties are uniformly regular for bounded partition problems.*

Proof. The proof is essentially the same as the proof for local subgraph properties. The only differences reflect the use of more finite state machines in the definition of local partition properties. Since no infinite quantities are introduced by these changes, the number of equivalence classes for the property will still be finite. □

Corollary 5.10. *There is an effective procedure to build the multiplication tables for any local bounded partition property for any k-terminal recursive family of graphs.*

Proof. The proof is identical to the proof of Theorem 4.8. □

6. General subgraph problems

Bern et al. [6] showed how to extend their results for vertex subset properties to subgraph properties involving edges. The same extension allows us to extend all of our results about vertex-induced subgraph properties to general subgraph properties. The basic idea of the extension is quite simple. Suppose that Γ is a k-terminal recursive family and that $\circ$ is a composition operation of Γ. When $\circ$ is extended to graph $\times$ subgraph pairs (G, H) for vertex-induced subgraph properties as in Definition 2.5, it is only necessary to remember which vertices of G are *included* in H because included vertices that are adjacent in G are also adjacent in H. For general subgraph properties, this is not necessarily the case, so the *included edges* must also be remembered. Of course, if an edge is included, its two endpoints must also be included, but the converse is not necessarily true. If an application of $\circ$ adds l new edges, then the included edges can be remembered by using 2^l corresponding extended composition operations for $\Gamma^\times$. This causes an increase in the number of extended composition operations for $\Gamma^\times$, but, since $l \leqslant k^2$, the increase is by a constant amount.

The definitions of homomorphism and regular property are not changed by the extension to general subgraph properties. The linear-time dynamic programming algorithm and the unsolvability result from Section 3 are also unchanged and the pumping lemma requires only obvious minor modifications.

For general subgraph properties, the definition of locality requires modification. The intuitive meaning of locality is unchanged: a property is local if it can be verified by examining a bounded neighbourhood of each vertex. For vertex-induced subgraphs, the truth value of a vertex property for a vertex v can be determined by examining the neighbours of v in G because it is implicit that two vertiex are adjacent in H iff they are both included and they are adjacent in G. For general subgraphs, the

definition of neighbourhood needs to be refined to reflect the fact that adjacency in H is not necessarily inherited from G. For an included vertex v, it is necessary to examine two types of neighbourhood: $N_H(v)$ is the set of all vertices that are adjacent to v in H, and $N_G(v)$ is the set of all vertices that are adjacent to v in G but not in H. For an excluded vertex v, the definition of $N(v)$ is unchanged because it is implicit that an excluded vertex cannot have an incident included edge. The only other change required to generalize Definition 4.1 to general subgraph properties is that six automata are now required in the inductive part of the definition instead of four. For included vertices, M_1^H and M_2^H are used to check $N_H(v)$, and M_1^G and M_2^G are used to check $N_G(v)$. For excluded vertices, M_3 and M_4 are used to check $N(v)$ as before.

Example 6.1. The problem of determining whether a graph contains a collection of disjoint cycles that includes all vertices of the graph is an example of a 1-local subgraph problem. The 0-local vertex property is true for all included vertices and false for all excluded vertices. The languages accepted by the six automata associated with the 1-local vertex property are $L(M_1^H) = \{2\}$, $L(M_2^H) = \{0\}$, $L(M_1^G) = \mathbb{Z}_0^+$, $L(M_2^G) = \{0\}$, and $L(M_3) = L(M_4) = \{\ \}$. Thus, the 1-local property is false for all excluded vertices and true only for included vertices that have exactly two neighbours in H. Using $L(M_a) = \mathbb{Z}_0^+$ and $L(M_b) = \{0\}$ ensures that every vertex is included.

The following two results are generalizations of Theorems 4.5 and 4.8 from vertex-induced subgraph properties to general subgraph properties. Since the only changes in the proofs result from the increased number of automata, the proofs are omitted.

Theorem 6.2. *All local subgraph properties are uniformly regular.*

Theorem 6.3. *There is an effective procedure to build the multiplication tables for any local property for any k-terminal recursive family of graphs.*

7. Discussion

We have shown that any optimal subgraph or vertex partition problem based on a property that is locally verifiable in constant time per vertex is regular with respect to all k-terminal recursive families of graphs. This implies the existence of a linear-time algorithm for any such problem (when the input is in the form of a parse tree) and we have given an effective procedure for constructing these algorithms. Although we have not proved if here, it should be obvious that our results can be extended to problems involving edge-induced bounded partitions by suitable changes to the definitions. We have also presented a pumping lemma which can be used to establish that any attempt to generalize our definition of locality by replacing the finite state automata with more powerful machines gives properties that are not uniformly regular. That such properties are difficult to work with is established by our proof that it is unsolvable to

compute the multiplication tables for an arbitrary property with respect to a particular k-terminal recursive family.

It is tempting to conclude that locality somehow "captures" the notion of uniform regularity, but this is not true. The Hamiltonian circuit problem is based on a uniformly regular property (Hamiltonicity) that is not local. Hamiltonicity is not local because any verification procedure that is restricted to examining a bounded neighbourhood of each vertex cannot distinguish between a Hamiltonian circuit and a union of disjoint circuits. Hamiltonicity is a uniformly regular property as a special case of the following more general result.

Theorem 7.1. *The property that a subgraph is a cycle is uniformly regular.*

Proof. Let Γ be a k-terminal recursive family of graphs and let $\circ$ be a composition operation of Γ. Consider a graph $\times$ subgraph pair $(G, H) \in \Gamma^{\times}$ where $G = (V, E)$. If there is an included vertex $v \in V$ with $|N_H(v)| > 2$, then (G, H) does not have the "cycle property" (i.e., H is not a cycle), and composition operations involving (G, H) will result in pairs that do not have the cycle property. Also, if $|N_H(v)| < 2$ for an included internal vertex v, then v can never be part of a cycle. We group all pairs which have such vertices into a rejecting equivalence class. Now, assume that $|N_H(v)| = 2$ for all included internal vertices and $|N_H(v)| \leqslant 2$ for all included terminals. If $|N_H(v)| = 2$ for all included terminals then H is either a cycle or a union of two or more disjoint cycles. We group all pairs of the first kind into the single accepting class and all pairs of the second kind into a rejecting class. If more edges are added to H when (G, H) is in the accepting class, then the resulting graph $\times$ subgraph pair will belong to a rejecting class. For each included terminal v of (G, H) with $|N_H(v)| = 1$ there must be another such terminal w such that there is a path from v to w in H. If all such pairs of terminals are known for each graph $\times$ subgraph pair, then it can be determined whether the result of a composition operation has the cycle property. Since the number of such pairs of terminals in a graph $\times$ subgraph pair is at most $k/2$, and this is a finite number, the number of equivalence classes needed to record the information to identify all of the pairs of terminals is also finite. □

Theorem 7.1 shows that locality, as we have defined it, does not completely characterize uniform regularity. The same can be said of three other notions of locality that have appeared in the literature. Locality as defined in our earlier work [12] is a weaker version of our current definition. Bodlaender's definitions of the class LCC and several subclasses of LCC [7] are similar to the definition in this paper, but the existence of polynomial algorithms of LCC problems is only established for input graphs with maximum degree bounded by a constant. (The existence of linear-time algorithms requires additional restrictions.) Seese's notion of *existential locally verifiable* is based on monadic second-order logic without universal quantification; however, Courcelle [9] has shown that all properties expressible in monadic second-order logic with universal quantification are uniformly regular.

It should be possible to extend our approach to obtain a complete characterization of uniform regularity. The only technical purposes served by restricting a vertex property p to being j-local is to bound the number of states in the finite state automaton that verifies p, and to permit relatively straightforward induction proofs. We could obtain a more general characterization of uniform regularity by replacing our condition that a finite state machine can verify that a vertex property p holds by examining its j-neighbourhood, by a condition that a finite state machine can verify that p holds for a vertex in time that is independent of the size of the input. (Of course, the time will depend on k which is fixed for any particular k-terminal recursive family.) This condition is consistent with our pumping lemma and is probably very close to a complete characterization of uniform regularity. Unfortunately, our simple locality-based test for uniform regularity would be lost in the process of this generalization.

We believe that the approach described in this paper is a useful complement to existing methodologies. Our heuristic for identifying uniformly regular properties is based on the graph theoretic concept of locality. We think that most researchers using graph algorithms will find this to be a simple and intuitive tool. If a property fails the locality test, then the more general condition of expressability in extended monadic second-order logic [3, 8, 9] can be used, or our pumping lemma can be used to try to show that the property is not uniformly regular. Development of an algorithm using our construction method is easy. It only requires the specification of a property in terms of a finite state automaton. Very simple automata are sufficient for specifying most properties. For example, the reader is invited to specify the automaton for finding a spanning Eulerian subgraph with no degree 2 vertices.

One final technical point concerning regularity should be made. We regard regularity as a structural property that permits the development of linear-time algorithms for optimization problems on k-terminal recursive families of graphs. A solution for an instance of one of these optimization problems must be both *feasible* (i.e., must satisfy a regular property) and *optimal* with respect to some objective function. For a table-based dynamic programming algorithm to run in linear time, it must be possible to evaluate the objective function in constant (average) time at each node of a parse tree. We consider the required restrictions on objective functions to be a separate issue and refer the reader to [3] for a discussion.

Acnowledgements

We thank the referees for their constructive criticism and the editors for their patience.

References

[1] S. Arnborg, Efficient algorithms for combinatorial problems on graphs with bounded decomposibility—a survey, BIT 25 (1985) 2–33.

[2] S. Arnborg, D.G. Corneil and A. Proskurowski, Complexity of finding embeddings in a k-tree, SIAM J. Algebraic Discrete Methods 8 (1987) 277–284.
[3] S. Arnborg, J. Lagergren and D. Seese, Problems easy for tree-decomposable graphs, J. Algorithms, submitted. (Extended abstract appears in: T. Lepisto and A. Salomaa, eds., Proceedings of the 15th International Colloquium on Automata, Languages and Programming (Tampere, Finland, 1988), Lecture Notes in Computer Science 317 (Springer, Berlin, 1988) 38–51.)
[4] S, Arnborg and A. Proskurowski, Characterization and recognition of partial 3-trees, SIAM J. Algebraic Discrete Methods 7 (1986) 305–314.
[5] S. Arnborg and A. Proskurowski, Linear time algorithms for NP-hard problems restricted to partial k-trees, Discrete Appl. Math. 23 (1989) 11–24.
[6] M.W. Bern, E.L. Lawler and A.L. Wong, Linear-time computation of optimal subgraphs of decomposable graphs, J. Algorithms 8 (1987) 216–235. (Preliminary version appeared as Why certain subgraph computations require only linear time, in: Proceedings of the 26th IEEE Symposium on Foundations of Computer Science (1985) 117–125.)
[7] H.L. Bodlaender, Dynamic programming on graphs with bounded tree-width, in: T. Lepisto and A. Salomaa, eds., Proceedings of the 15th International Colloquium on Automata, Languagues and Programming (Tampere, Finland, 1988), Lecture Notes in Computer Science 317 (Springer, Berlin, 1988) 105–118. (Also see Tech. Report RUU-CS-87-22, Department of Computer Science, University of Utrecht, Netherlands (1987).)
[8] R.B. Borie, R.G. Parker and C.A. Tovey, Automatic generation of linear algorithms from predicate calculus descriptions of problems of recursively constructed graph families, manuscript, Georgia Institute of Technology (1988).
[9] B. Courcelle, the monadic second-order logic of graphs I: Recognizable sets of finite graphs, Inform. and Comput. 85 (1990) 12–75.
[10] S.T. Hedetniemi, Bibliography of algorithms on partial k-trees, manuscript (1989).
[11] J.E. Hopcroft and J.D. Ullman, Introduction to Automata Theory, Languanges, and Computation, Ch. 8 (Addison-Wesley, Reading, MA, 1979).
[12] S. Mahajan and J.G. Peters, Algorithms for regular properties in recursive graphs, in: Proceedings of the 25th Allerton Conference on Communication, Control, and Computing (1987) 14–23. (Also see Tech. Report TR 87-7, School of Computing Science, Simon Fraser University (1987).)
[13] D. Seese, Tree-partite graphs and the complexity of algorithms, in: L. Budech, ed., Fundamental of Computing Theory, Proceedings, Lecture Notes in Computer Science 199 (Springer, Berlin, 1985) 412–421.
[14] K. Takamizawa, T. Nishizeki and N. Saito, Linear-time computability of combinatorial problems on series–parallel graphs, J. ACM 29 (1982) 623–641.
[15] T.V. Wimer, Linear algorithms on k-terminal graphs, Ph.D. Thesis, Clemson University, Clemson, SC (1987).
[16] T.V. Wimer, S.T. Hedetniemi and R. Laskar, A methodology for constructing linear graph algorithms, Congr. Numer. 50 (1985) 43–60.